E-MAIL CERTIFICATION SERVICE

BACKGROUND

[0001] The present patent application is in the field of electronic mail and, more particularly, is directed to verification, by a receiver of an e-mail message, of the integrity of the received e-mail message.

[0002] Electronic mail has become a useful tool in our personal and business lives. Unfortunately, like other useful technologies, it has also become an intrusion. For example, only the most diligent among us are able to avoid scurrilous attempts to sell us nutritional supplements, mortgages, pornography and numerous other "products." More recently, scammers have taken to using "spoof" e-mails in an illegitimate attempt to gain access to our personal information. For example, such scammers have spoofed BestBuy and eBay, attempting to entice unsuspecting users into providing personal information such as social security numbers and credit card numbers. Thus, as useful as e-mail has proven to be in our lives, it can be dangerous to blindly assume that received e-mail is legitimate. However, efforts at minimizing intrusion of technologies often unavoidably diminish the usefulness of the technology whose intrusion we are seeking to minimize.

[0003] For example, "spam" catchers typically catch legitimate e-mail messages in addition to spam e-mail messages. Users must carefully scrutinize the caught e-mail messages, lest any of them be legitimate and unintentionally ignored. Also, spoof e-mail messages are more difficult to detect, as they appear in many respects to be legitimate.

[0004] There have been a number of attempts to address the concerns with e-mail. One notorious attempt is described in USP 5,999,967, to Sundsted. Sundsted has proposed attaching an "electronic stamp" to each e-mail message sent, where the receiver of the e-mail message receives money from the sender. The receiver can determine whether it is "worth it" (from the value of the attached stamp) to read the e-mail and receive the money. Because Sundsted employs "stamps" having monetary value associated with them, there is a practical requirement (which is difficult to achieve) that the system to exchange value be secured against fraud. Furthermore, even if the system to exchange value can be made secure, there is nothing that allows a receiver of e-mail to discriminate between senders from whom it is desirable to receive e-mail and senders from who it is undesirable to receive e-mail apart from the monetary benefit to the

receiver who reads e-mail. Perhaps even more significantly, nothing in the electronic stamp allows one to assess the integrity of the e-mail.

[0005] In many respects, a proposed system known as "HashCash" is similar to the system described in the Sundsted disclosure. The proposed HashCash system is such that, before an e-mail message is sent, a significant particular math computation must be performed on the sending computer to generate a token. This computation is such that, for example, it would take up to 15 seconds on a standard 1 GHz PC. The token is incorporated into the e-mail message. The receiving computer performs a relatively simple computation to verify that the token is, in fact, the result of the particular a math computation performed on the sending computer. A drawback of HashCash, then, is that anyone who is willing to undergo the computational burden can send e-mail messages unimpeded. That is, like the system described in the Sundsted patent, there is nothing in the token that allows the receiving side to discriminate between senders from whom it is desirable to receive e-mail and senders from whom it is undesirable to receive e-mail, beyond verifying that the sender did, in fact, incur the computational expense to generate the HashCash token. That is, in some sense, HashCash merely substitutes computational expense for the monetary expense of the Sundsted system (albeit there is money or other value received by the e-mail recipient).

BRIEF SUMMARY

[0006] A method is provided to handle an electronic mail message such that the receiver of the e-mail message can verify the integrity of the message. A request is provided from a sender's side to a service. The request includes information regarding the e-mail message. The service processes at least a portion of the request to generate a result. For example, the service may encrypt the portion of the request, according to a public/private key encryption scheme, to generate a digital signature as the result. The service provides the result to the sender's side.

[0007] At the sender's side, the result is incorporated into the e-mail message and the result-incorporated message is transmitted via an e-mail system. At the receiver's side, the result-incorporated e-mail message is processed to assess the integrity of the received e-mail message.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a flowchart illustrating a process in accordance with an aspect;

FIG. 2 schematically illustrates the relationship between various entities as the Figure 1 method is executed;

FIG. 3 is a flowchart illustrating, in greater detail, a portion of the Figure 1 method that is executed at the receiver's side.

FIG. 4 illustrates a portion of the Figure 3 method where it is determined if a received e-mail message is a result-incorporated message and appropriate action is taken.

FIG. 5 illustrates the processing of the service being carried out on a plurality of servers.

FIG. 6 illustrates an example of how public keys are made available to e-mail client programs at a sender's side (which will later be a receiver's side).

FIG. 7 illustrates an example of processing at a receiver's side to cause invitations to be sent to a reply-to address.

FIG. 8 illustrates an example of interaction between a browser program, an e-mail program and a service to compose and send content.

FIG. 9 illustrates an example of interaction between an e-mail client and a service for receiving information from the service regarding an e-mail sender.

FIG. 10 illustrates an example mechanism to synchronize information between a client and a server.

FIG. 11 illustrates an example of secure communication between a client computer and a service.

DETAILED DESCRIPTION

[0008] Figure 1 illustrates a broad aspect of a method/system to handle an e-mail message such that a receiver of the e-mail message can assess the integrity of the e-mail message. Figure 2 illustrates the architecture of the method/system. Referring to Figures 1 and 2, a plurality of senders 102a through 102n (referred to generically in Figure 1 by the reference numeral 102) and a plurality of receivers 106a through 106n (referred to generically in Figure 1 by the reference numeral 106) are, in a basic form, present in any e-mail system. That is, any sender 102 can send an e-mail message to any receiver 106 via the e-mail system 118.

[0009] Referring specifically to Figure 1, at step 110, an e-mail is created at the sender's side. 102 For example, the user authors an original e-mail message using standard e-mail client software (e.g., Outlook 2000 or Outlook 2003), including indicating a desired recipient of the message. At step 112, information regarding the e-mail is provided to the service 104. The user does not need to take any special course of action to cause the information to be provided to the service 104. The process is intentionally

"transparent" to the user to minimize possible confusion and extra steps that might otherwise interfere with the rapid deployment and use of the service 104.

[0010] For example, upon detecting an e-mail message to be sent, programming code either embedded within the email client or included as an add-on component intercepts the e-mail message. For example, many of Microsoft's Outlook products include a facility for third party developers to provide plug-in code to enhance the functionality of the Outlook e-mail client program. Other e-mail client programs do not explicitly provide a convenient facility for third-party developers to provide such plug-in code, but the plug-in code can be provided nonetheless.

[0011] The mail client interacts with the service 104, in the form of a single TCP/IP request using a standard Internet protocol such as HTTP or HTTPS. By using standard Internet protocols for communication with the service 104, packets of the communication will generally not be blocked by corporate or home DSL firewalls. Packets transmitted by HTTPS are also generally immune from network snooping since HTTPS is a secure protocol. It is noted that payload data of HTTP transmissions may be internally encrypted.

[0012] In one specific example, programming at the sender's side determines the following information regarding the e-mail message:

- A digest. The digest is created by processing the true text of the e-mail body into a mathematical representation that functions as a fingerprint of sort, yet at the same time prevents disclosure (leaking) of the actual text --- ensuring privacy for the author, yet allowing us to detect changes. From the digest, it can be determined if a change took place, even without knowing what was the original text was or how it was changed.
- A random salt to be associated with this digital stamp – 12 bytes. The use of the salt is discussed in greater detail later.
- Digests for each email recipient, which incorporate the salt. Specifically, the salt is appended to a recipient address, and the SHA1 digest is computed on the combined structure, which results in a 20-byte digest. The least significant 12 bytes of that binary digest is transformed to a Base64 ASCII representation (16 characters long) which is embedded into the stamp. This process is performed for each recipient address.

In general, the information regarding the e-mail message includes information such that, after being processed by the service 104 to generate a result, as discussed below, the receiver's side 106 can process the result and assess the integrity of the e-mail message.

[0013] In some embodiments, the information provided to the service 104 includes the sender's identity and authentication (typically account/password). This information, as opposed to being information regarding the e-mail message, can be characterized as information about the sender, from which the service 104 can authenticate the sender.

[0014] As discussed above, it is not necessary for the sender's side 102 to provide the actual content (body) of the e-mail message to the service 104. This contributes to maintaining the privacy, security and comfort of the sending user. The e-mail message body is "hashed" by the mail client at the sender's side 102 to derive a multi-digit numerical sequence which represents the substance of the message without disclosing its content. A variety of digital hashing/digest techniques could be employed, such as SHA-1. The hash result, more commonly known as a message digest, is a mathematically unique number generated according to a particular algorithm. The theory of the industry standard SHA-1 algorithm is that it will never generate the same message digest for non-identical input text, yet it will always generate the same digest for the identical input text. (Algorithms other than SHA-1 and SHA-1 like algorithms are particularly discussed later.) Message digests are most commonly used in security schemes utilizing RSA encryption systems and have been accepted by industry for many years as a very dependable form of representing messages and for detecting unauthorized changes to such messages. It is the message digest, not the substance of the message itself, that is the information regarding the e-mail message provided from the sender's side 102 to the service 104.

[0015] Referring still to Figure 1, at step 114, the service 104 processes the information regarding the e-mail to generate a result. In particular, the information regarding the e-mail is routed, using load-balancing techniques, to an appropriate HTTP server of the service 104. A goal of load balancing is to reduce load or bottlenecks and minimize risk due to system failures. Off-the-shelf hardware contributes to this. Also, senders may be "assigned" to particular groups, and the groups may be assigned to sets of servers. In this instance, each sender will locally save the URL/IP for its associated group and, thus, will be able to help avoid traffic jams by going directly to a server that is able to handle its requests. This load can be

spread around the country/world so the "whole universe" is not potentially trying to simultaneously hit one single access point. Downtime is improved since, if a server does go down, only its assigned group is affected.

[0016] In some examples, a physical IP address will not be returned, since the IP address is subject to change often but, rather, "routing information" is provided. The routing information in one example is comprised of a group reference and a record number within the group. When the request arrives, the front-line server queries a small locally-cached table that associates groups with IP addresses – and subsequently routes the request to the designated IP. This results in a simple level of indirection that is mostly immune from changes that may be made in the service 104, and also does not leak IP targets for the middle tier servers.

[0017] In some examples, if it is determined that the server accessed by a particular sender is inappropriate for some reason, then the service determines a different server of the service 104 to which the sender's request is redirected. The URL/IP or other routing information for the different server is provided back to the sender for it to locally save for use when subsequent requests are sent to the service 104 by that sender.

[0018] For example, Figure 5 illustrates the processing of the service being carried out on a plurality of servers 502a through 502c. While three servers 502 are illustrated, typically many more would be employed. The e-mail client software of each sender, as part of the information provided to the service 104, provides an indication of the particular server that is to process the request. The Figure 5 example illustrates that the indication is a URL (e.g., http://service.1stcertified.com/00001) associated with that particular server, although other indications may be employed.

[0019] If the service 104 determines that it has been provided an incorrect indication by the e-mail client software, then the service 104 directs the information to the proper server 502. For example, an incorrect indication may be a non-existent URL or may be a URL that corresponds to a server that is actually not assigned to service the particular requesting e-mail client. In both cases, such requests may be handled initially by a default server 504 of the service 104, reachable via the URL http://service.1stcertified.com/default, and the default server 504 determines which server 502 is the proper server and directs the information to that proper server to be processed.

[0020] When the result is provided back to the e-mail client software from the proper server, an indication of the proper server 502 is also provided back to the sender's side. Thereafter, when the sender's side provides information to the service 104, it will provide the newly-provided indication of the server 502 to the service 104. as part of subsequent information provided to the service 104. Thus, operational latency is reduced and potentially expensive real-time lookup is avoided as requests enter the service 104 (to find out which server handles a given user's account). In addition, bottlenecks are minimized on the lookup server.

[0021] Referring again to Figure 1, the service 104 executes a program to authenticate the account name and password. In addition, the sender's e-mail address, recipient's e-mail addresses and message body digest are combined (e.g., concatenated) along with additional housekeeping information (timestamp, sequence numbers, priority, sorting, keywords, etc.) and provided as input to the industry-standard SHA-1 algorithm, to form a 160-bit unique result. In some embodiments, a portion of the 160-bit unique result is discarded in order to increase performance and reduce the size of the result without meaningfully degrading the ability of the receiving side 106 to adequately assess the integrity of the e-mail message.

[0022] Still at step 114, the service 104 signs the result (entire, or a portion thereof as discussed above) of the SHA-1 algorithm with a private key, such that it can be decoded using the corresponding public key. The private key is kept secret by the service 104, and the public key is made available to the receiver side 106. In some embodiments, the public key is "embedded" within the receiver side e-mail client software. The signing determines a sequence result that is an alphanumeric sequence of characters approximately 128 characters long. The service-determined sequence result is returned from the service 104 to the requesting sender 102. In some embodiments, a corresponding result code is also provided to the sender 102.

[0023] In one specific example, the service 104 prepares a data structure comprised of information provided in the client request along with other information which can either be computed on the fly or obtained from the sender's account profile. Theoretically, most of the information contained within a digital stamp need not be kept secret. Due to the way the RSA encryption scheme operates, there is an approximately 128-byte section of the stamp which contains some payload – which is RSA encrypted. 81 bytes of this area (at present) are reserved for variable payload data. In some examples, all of the payload fits into the 81 bytes. In other examples, the

payload overflows into another area, and a digest is computed on the overflow area and that digest is embedded into the RSA encrypted space, to guard against tampering. The RSA space is encrypted with the private key associated with the service 104 (this is the signing process) so that the client software at the receiving sides 106 can decode the digests with the public key. Many types of information may be provided in the "stamp" result. In general, the result is a "signed" payload area to convey information. For example, information that may be provided as payload may include:

- information such as the sender/recipient addresses, and body digest, used at the receiver side 106 to assess the integrity of the received e-mail
- Fraud detection – e.g., sequence numbers
- Sender profile information. Information about the sender to make available to the recipient, or include for the express purpose of facilitating subsequent verbose profile requests by the recipient about the sender. This may even include, for example, an icon reference to put a branded icon next to this email item.

In the specific example, the stamp returned to the sender's side 102 is in an ASCII format that includes versioning and other housekeeping information to facilitate decoding by e-mail clients at the receiver sides 106.

[0024] In one specific example, the general format of a stamp is as follows (Base64 fields URL-encoded):

```
v={version}&c={serialno}&e={envelope}[&d={payload}]
```

The v is the numeric version reference for the stamp. The initial value is the character "1". This is a high-level version indicator and will always be the very first parameter present in the stamp string. Any software tasked with interpreting stamps, such as the Outlook Add-in (client-side programming) at the receiver's side 106, inspects this value before looking at any other parameters. Deep inspection of the remaining parameters should be attempted only if the version is recognized. It is possible and likely that other version identifiers may appear within other components of a digital stamp. If such other identifiers are present, they should be interpreted as however appropriate. What is noteworthy at this level is that the version identifier indicates which additional parameters should be expected and how they should be interpreted.

[0025] In the default case of v=1, the c, e parameters are expected to be present and should be interpreted as specified herein. The d parameter is optional depending on the contents of the envelope (e).

[0026] The c parameter identifies the serial number of the 1stCertified private RSA key certificate used to sign/encrypt the envelope (e). Valid values are integers from 1 to 999999. No commas, no leading zeros. Although the initial value will likely be "1", this is a volatile parameter that can be expected to change frequently. Note that in some other places where certificate SNs are referenced a 6-digit value with leading zeros is specified – be sure not to confuse these different requirements. Certificates are typically cached in the registry by the client-side add in programming and retrieved by serial number to decrypt and permit the inspection of digital stamps. In the event an unknown certificate is referenced, the low level code within the client-side add-in programming is responsible for obtaining the requisite certificate from the datacenter in real time.

[0027] The e parameter represents the RSA envelope. Formally, the value of this parameter is the Base64 string output obtained from transforming the binary envelope structure. More on this below.

[0028] The d parameter is optionally present and holds additional payload data. The payload field can hold any essential data that could not otherwise fit inside the envelope. Payload data need not be encrypted since it does not contain any information that is required to be kept secret. A flag within the envelope determines whether or not the d parameter is required and whether or not it is encrypted.

[0029] The envelope for v=1 is transported as a Base64 representation of a 1024-bit RSA encryption result. The parameter text must therefore first be transformed back to its binary state (128 bytes). Next, the byte[128] array is decrypted using the RSA public key corresponding to the certificate serial number referenced in the c parameter.

[0030] The envelope has the following structure:

```
        Word CheckPattern            // always 55AAH
        Word EnvelopeVersion              // initially 1
        Word MailClass                    // enum, see table
        Byte[20] BodyDigest               // Fingerprint of
email body
        Word BodyDigestMethod             // enum, indicates
method used
```

```
    Byte PayloadFormat        // enum, 0= data in union
                              // 1= 'd' parameter
                          required
    Payload Union {                  // 80 bytes max
         Mode A // PayloadFormat = 0
         {
              Byte[81] Payload     // additional
parameters
         }
         Mode B // PayloadFormat = 1
         {
              Word Algorithm; // 0=AES128CBC (default),
                              // 1=3DES128CBC,
                          2=3DES192CBC,
                              // 3=CLEARTEXT
              Byte[20] DataDigest       // SHA1 digest of
'd' parameter
              Byte[24] SessionKey // AES
encryption/session key
                                   // key always filled
in even
                                   // if not needed
         }
    }
```

[0031] The CheckPattern field is provided to allow software to determine that the envelope has been correctly decrypted. Bad/wrong decryption certificates and envelope data manipulation are easily detected. The value of this field will always be 55AAH.

[0032] The EnvelopeVersion field specifies the internal format of the envelope. After decrypting, software must inspect this value to determine that it is capable of interpreting this structure.

[0033] The MailClass field is an enumeration representing the various classes of mail service offered by 1stCertified. Generally, the class of mail relates directly to which folder within the 1stCertified InBox that incoming mail is deposited into. An exception

to such may be when the user has deleted a folder (or it may be added back in automatically for required folders).

[0034] The MailClass can be understood from the following table:

0	UNKNOWN	This is not a valid mail class. This value will not be found in valid envelopes.
1	PERSONAL	Person to person emails. This is the class used when users send certified email using Outlook. This class of mail cannot be disabled.
2	REQUESTED	Refers to user-requested commercial email on topics selected and maintained by the user in their online profile. This class of mail cannot be disabled; however, users can restrict mail from specified senders or for specific topics.
3	COUPON	Coupon-oriented offers based on topics selected and maintained by the user in their online profile. User's may disable reception of this class of mail as a whole, or for selected senders/merchants/topics.
4	COMMUNITY	Emails that originate from senders/merchants within the local area. The radius for the local area is specified by the user in their profile. Users may disable reception of this class of mail as a whole, or for selected senders.
5	MYCOMPANY	*Not an official mail class* – no stamps will created with this class.
6	NEWSLETTER	Email is an online newsletter to which the user has subscribed. The exact use of this class is not yet figured out … but class is reserved.

[0035] The names for the Outlook folders created for the various classes of mail are internationalized. If the Outlook Add-in receives an email with an unrecognized mail class, it should "pretend" the email is PERSONAL and act accordingly. Of course, this would likely mean that the Add-in is not the most current version.

[0036] The BodyDigest field contains the fingerprint of the email body that was computed by the sender. This is an SHA1 digest of "selected" portions of the body. See Body Fingerprint Specification for more details on this. The BodyDigestMethod field is an enumeration value indicating which method (in case we have several) was used to compute the fingerprint. Initially, this value will be 0, indicating the default method. It is not yet clear or know when, if or why we might employ multiple methods. Just planning ahead.

[0037] The PayloadFormat field is an enumeration that identifies which mode of the Union is utilized. If the value is 0, then the union is interpreted as a null terminated text string. The d parameter is not used and need not be present. If the value is 1, the digest contained within the union holds the SHA1 digest for the d parameter – and the d parameter is therefore required to be present. If encryption is required, then in addition the d parameter is encrypted using the specified AES encryption key. In this case, the d parameter is the Base64 transform of the encryption result, and appropriate steps must be taken to convert to binary, decrypt, and arrive at the clear text, and finally check the digest. Allowed values for the Algorithm are CLEARTEXT(3) and AES128CBC(0). The session key is always filled to 24 bytes regardless if actually used. Further, the entire union should be padded with random bytes to prevent long runs of zeros in the stream.

[0038] The use and format of the payload data is as follows: If the payload data happens to be short enough to fit in the envelope, then only the envelope is used. However, if the payload data is longer, then a way is provided to include either clear or encrypted data outside of the envelope. Both clear/encrypted data (d) are validated with a digest. Irrespective of the location of the payload data, it always takes the form of a text string in HTTP POST format – URL escaped, etc.

[0039] The reason the payload data is variable in size is that the sender email address is kept track of, as well as all normal and CC recipients. Although the BCC recipients may be accessible, it would pose a privacy issue to include them in the payload, even if encrypted. A snooper need only use the public key and crack the payload, exposing the list of BCC recipients. Therefore, it is a design decision to not include BCC addresses. In many cases, whereby there is only a single sender and a single recipient, such will likely all fit within the envelope.

[0040] Code implementing this payload feature first constructs the full payload parameter string and determine its length. If small enough to fit within the envelope,

the envelope only is used. Otherwise, the entire string is placed in the d parameter and the associated digest is placed into the envelope. In some examples, the d parameter may not be encrypted, but it is recommended that client code should fully implement and test this (encryption) capability since the service 104 may switch modes with know notice.

[0041] Since RSA operations work on 1024-bit (128-byte) chunks, there is no performance penalty for putting as much information as will fit within the envelope. Further, processing time can be saved both at the service 104 and at the receiving sides 106 by not needing to deal with digests/encryption for d payloads. Thus, flexibility has been designed into the stamp/envelope format.

[0042] The payload string has the following format (fields url-encoded):

```
a={accountID}&tr={n}&s={sender}&t={recipient_list}[&r={re
plyto}]
        [&ex={expires}&gmt={0|1}]
        [&id={GUID}]
        &seq={number}
        [&icon={icon reference}]
        &salt={salt value provided by client}
```

[0043] The a parameter is the sender's 1stCertified numeric AccountID (PkID).

[0044] The tr parameter indicates the trusted identity level of the sender. Valid integer values are:

1-Anonymous

2-Verified

3-Organization

[0045] These values are consistent throughout. 0-Unknown should not be possible in a stamp.

[0046] The s parameter specifies the sender. This should normally match the RFC822 "From" header email address, but may not for malformed/spoofed emails. The sender's email address is one that has been verified, in that an email has been sent to that address and a required click-back has been performed in order to affirm the user is in control of that address. This value will be exposed as a property from the high-level validation COM object.

[0047] The optional r parameter is the RFC822 "ReplyTo" email address. This is optional in the sense that if missing, it is assumed to be the same as the s parameter. This value is exposed as a property in high-level object also. In some examples, to save space in the payload, if C++ code preparing stamp sees that the ReplyTo is the same as the Sender email, then the ReplyTo is not placed into the payload, and the ReplyTo is treated later as being the same as Sender.

[0048] The t parameter is a semicolon-separated list of To/CC recipients. Instead of ASCII text, addresses are represented as Base64 partial salted digests. This includes both normal recipients and CC recipients. This value will be exposed as a collection from the high-level validation COM object. The listed recipients should match the RFC822 "TO" and "CC" headers under normal circumstances. Note that only email addresses are listed here – not plain text names as allowed in RFC822 headers.

[0049] The optional ex parameter indicates when an email expires. This parameter is used in some special circumstances, such as to process expiring coupons. The gmt parameter is used whenever ex is present and indicates that the time is GMT if 1, or local time if 0.

[0050] The optional id parameter indicates a GUID or other similar alphanumeric string which uniquely represents the email. This parameter is intended to be used for commercial (requested/coupons) emails sent directly as part of an email advertising campaign. This identifier is used by other APIs as a unique reference to a piece of mail. The reference will associate a recipient to an advertiser to a campaign and will therefore facilitate the opt-out process. In general, it should only be assumed that, whatever string is returned here will be accepted by the service when a reference to a commercial email is required. In some examples, the string contains sub-fields to help the service operate more efficiently (i.e., faster).

[0051] The seq parameter represents a statistically unique 32-bit integer value to be associated with this digital stamp, such that when combined with the AccountID, represents a globally unique identifier amongst all digital stamps across all accounts. It is likely, but not guaranteed, that this value will be generated simply by incrementing a value within the user's account each time a stamp is issued. The exact requirement is that it be statistically unique. The purpose served by this parameter is to help prevent hackers from reusing stamps by having the Outlook client code cache the AccountID:Sequence values of recently validated stamps – thereby facilitating the detection of a duplicate stamp (which should promptly be rejected as invalid).

[0052] It is noted that, just because a stamp has not been tampered with and appears to be correct, it is not necessarily completely valid for the current user having received it. The problem stems from the fact that RFC821 allows mail to be sent to potentially any mailbox, and the RFC822 "TO" header is not inspected nor verified as part of this process. This opens a hole whereby a rogue sender could obtain a valid stamp referencing their account and some don't-care recipient address, then send that email to thousands of mailboxes per RFC821. The client-side add-in programming will determine that the stamp has not been altered and is therefore valid – but, the caveat is that it is only valid for the "listed" recipients. As such, it becomes necessary for the Outlook Add-in to then determine if the recipient now attempting to validate an incoming email is satisfied they are one of the intended recipients. The Add-in would, for example, keep a list in the registry of email addresses the user is known by. Some users may have dozens of addresses they accept, and not all otherwise known to Outlook. For example, a user "Peter Smith" may have a single Outlook profile and corresponding POP3 account, but may receive mail for psmith@abc.com, sales@abc.com, webmaster@abc.com , support@abc.com, etc. It follows then that each of these addresses will be on the "approved" receiving address list so the Outlook Add-in will be able to subsequently determine that arriving e-mail is acceptable.

[0053] In cases within a digital stamp that HTTP POST formatted strings are used, URL-escaping is used.

[0054] An example of a "stamp" is shown below:

v=1&c=5&e=fJXdWRuEDFyIW40s0CRlXN%2bBp1qlBkBl1fCEGNX5DGlTtNIwL4Vq1lU1kqJn
%2bwK5TXVEKa4OufthbJi0pwsBEWm2LX2LlRkJ2miTrXjUxrU1AgZlm2xtw0hhyrkGlLr0fON%
2fKLSXcyKXNuBXw80MzGw63vphgW89lQzLCwR3t8A%3d&d=a%3d123458%26tr%3d2%2
6s%3dpsmith%2540pcdynamics.com%26t%3dtest%2540mytest.com%26seq%3d48299

The same stamp decoded is:

Version: 1
Size: 286 bytes
Certificate SN: 000005
External Payload: Yes (71 bytes)
Envelope Version: 1
Mail Class: PERSONAL
Body Digest Method: DEFAULT
Body Digest: 8C-B2-23-7D-06-79-CA-88-DB-64-64-EA-C6-0D-A9-63-45-51-39-64
Payload Format: EXTERNAL

Payload Algorithm: CLEARTEXT
AccountID: 123458
TrustedIdentity: 2
Sender Email: psmith@pcdynamics.com
Reply To:
Recipient List: test@mytest.com
Sequence Number: 48299
--end--

Tests to validate a stamp at the receiver's side, discussed more generally later, may specifically include (referring to specific fields in the example stamp):

- Make sure the *SenderEmail* is the same as the *FROM* mail header. Note that this is in relation to the actual email address portion of the header, which might also include some text name.
- Make sure the *SenderReplyTo* is the same as the *REPLYTO* mail header. Note that this is in relation to the actual email address portion of the header, which might also include some text name.
- Make sure the body fingerprints match by inspecting the *BodyFingerprintMatch* property.
- Make sure that the instant user is actually on the *Recipients* list. Users can go by many email addresses. A match or hit must exist between the recipient addresses listed in the stamp and the user's proclaimed *My Email Addresses* list.
- Test for replay attempts on digital stamp using *AccountID:SequenceNumber* cache lookup.

[0055] Further, in some embodiments, the service 104 performs a variety of accounting, logging and account management procedures at step 114 so that usage and quality of service can be monitored and so that, for example, billing functions can be executed as appropriate.

[0056] At step 116, the sender 102 incorporates the service-determined sequence result into the e-mail message. In one embodiment, the service-determined sequence result is incorporated into the e-mail as an SMTP X- header. Mail headers are commonly used within existing SMTP systems, and such systems include processes to perform the routing and housekeeping out of view of the users such that e-mail messages ultimately arrive in the inbox of their intended recipients. Industry-standard guidelines cover the

use of such mail headers, so e-mail messages including the mail headers are allowed to pass through the existing SMTP infrastructure without being blocked, without interfering with the quality of service delivered to users, and without being altered.

[0057] Turning back to Figures 1 and 2, the result-incorporated e-mail message is then transmitted via the e-mail system 118 to the sender's SMTP server, and, ultimately is forwarded to the designated recipient's POP3 email account. SMTP routing software along the way is free to either ignore or interpret the SMTP mail header holding the service-determined sequence result. Generally, the SMTP mail header will be ignored (with the exception of the SMTP server or e-mail client of the intended recipient).

[0058] The e-mail client software of some intended recipients may not be "enabled" (either does not have the capability or is not so configured) to process the SMTP mail header holding the service-determined sequence result. The sender need not know (or care) if the recipient is using enabled e-mail client software. If the e-mail client software of the receiver is enabled, the SMTP mail header holding the service-determined result will be processed. Otherwise, non-enabled receiver clients ignore the SMTP mail header, generally behaving as if the SMTP mail header was not included in the received e-mail message at all.

[0059] If the e-mail client software of the recipient is enabled, at the receiver side 106, the sequence result is processed at step 120 to assess the integrity of the received e-mail message. The e-mail message is received "normally" by the email client software from the recipient's SMTP/POP3 server. Before presenting the incoming e-mail message to the user, the e-mail message is preprocessed to assess the integrity of the received e-mail message. This may be accomplished by validation code that is, for example, embedded into the e-mail client; a third-party add-on component to the email client; integrated into an embedded or third-party anti-SPAM product; integrated in whole or in part, or as an add-on, within an advanced SMTP server such as Microsoft's Exchange Server.

[0060] Figure 3 illustrates, in greater detail, processing of step 120 at the receiver side 106. Turning now to Figure 3, upon interception of the incoming message, at step 302, the validation code generates a first SHA-1 result (or, alternately, a "fuzzy" digest, as discussed later), based on the body of the received e-mail (the part the user generally sees). If the body of the received e-mail has not materially changed since being composed by the sender 102 (more properly, since the message digest was created at step 112 of Figure 1), then the SHA-1digest result, computed at the receiver's side 106

in step 302, will be the same as the digest computed at the sender's side 102 and provided to the service 104 at step 112.

[0061] At step 304, the first SHA-1 result is concatenated (or otherwise grouped) with the sender's and recipient's e-mail addresses as contained in the received e-mail. This grouping is the basis of a second SHA-1 result generated at step 304. The second SHA-1 result nominally replicates the service-determined result (step 114 of Figure 1) computed by the service 104 and incorporated into the e-mail message at the sender's side 102 (step 116 of Figure 1). In some examples, the process of signing includes creating a SHA-1 digest of the material information, then encrypting the digest with a private key. In some examples, the material information may be encrypted. The digest is used to determine whether the information that has been effectively "notarized" is subsequently changed. One of the pieces of material information is the digest calculation result on the email body. Therefore, it can be determined that both the body has not changed, and that the other information has not changed. In some examples, the stamp format has a variable set of fields that allows for future growth – for example, icon references may be embedded in the stamp (to paying clients) so that their icon/logo shows next to mail received.

[0062] Also at step 306, the public key associated with the service 104 is used to decode the service-determined result incorporated into the received e-mail message, to determine a decoded service-determined result. If there has not been spoofing or hacking or other alteration of the e-mail message (whether willful or otherwise), the decoded service-determined result matches the second SHA-1 result. Once the integrity of the received e-mail has been assessed, appropriate action is taken. At step 308, the decoded service-determined result is compared to the second SHA-1digest result. If these are the same, then the received e-mail is "sound" and appropriate action is taken at step 310. If these are not the same, then the received e-mail is not "sound" and appropriate action is taken at step 312. An example of actions taken for validation is discussed above.

[0063] In one specific example, the validation process at the receiver's side 106 includes the following:

- Check stamp version info as a sanity test.
- Compute body digest in similar manner to when email was authored.
- Decode the RSA area of the digital stamp that exposes the structures described in the stamp specification. In the case where the overflow

(external) payload area was used, compute a digest on that area and compare to the digest stored in the RSA area. Determine if the digests match and, if not, indicate an error.

- o Compare the body digest computed at the receiver's side with the body digest saved in the RSA area of the stamp. Determine if the digests match and, if not, indicate an error.
- o Perform a matrix match test between the recipient addresses embedded in the stamp, and any addresses by which this user goes. There should be at least one match. If only partial digests are embedded in the stamp, partial salted digests of any addresses (aliases) the recipient goes by are created, and these are each checked against each of the digests within the stamp. If none match, then indicate an error. Otherwise, upon a single match, no others need be checked.
- o Perform other fraud tests such as making sure the ReplyTo in the stamp matches the one contained in the email. Do the same for the sender (From header).
- o Get the sequence number from the stamp and look up in a locally cached file to make sure that this same sequence number has not been recently used – prevents reply attacks whereby senders use programming to attempt to use the same stamp over and over again.

If the stamp appears to be "good," data is extracted from some fields of the stamp and inserted into the message record for the email (in the e-mail client). This allows for somewhat fast retrieval of properties/attributes within the user interface without needing to revalidate the stamp (which is a relatively CPU-intensive step). Such properties include the trust level of the sender, mail class (personal, newsletter, coupon, etc.).

[0064] In accordance with some embodiments, at the receiver's side 106, it is determined whether a received e-mail message is identical to e-mail messages previously received. This is typically a result of an identical e-mail message being sent to the recipient repeatedly. In specific examples, a database is maintained at the receiver's side 106 of service-determined results. By comparing the service-determined result associated with a received e-mail message to entries in the cache, it can be determined that the received e-mail message is identical to one or more e-mail messages previously received, and appropriate action can be taken.. In some

examples, to save disk space and increase efficiency of execution, each service-supplied result contains an embedded sequence number – so the sequence number is extracted from the result, associated with the sender's ID (also in the stamp) and only that set of information is saved in the replay cache. The entire server-supplied result need not be saved.

[0065] As discussed above, in accordance with some embodiments, in addition to the service-determined result, the SMTP message header includes additional information provided at the sender side 102. For example, this additional information may include keywords, which can be used by the e-mail client software at the receiver's side 106. This additional information can be processed and appropriate action taken. For example, the e-mail client software at the receiver's side 106 may use the information for sorting of e-mail messages or for otherwise controlling the placement of e-mail messages display in the e-mail inbox.

[0066] In some examples, processing at step 310 includes observing how the e-mail client at the receiver's side 106 reacts to (interacts with) the received result-incorporated e-mail message based on user input (see box 122 in Figure 1). For example, the user may indicate senders of e-mail messages that are to be on a blocked address list. Each time the e-mail client at the receiver's side 106 blocks a result-incorporated e-mail message, information regarding the blocking is provided back to the service 104. Other information regarding the user's interaction with the e-mail message may be included such as, for example, "click through" on links in the result-incorporated e-mail message. The service 104 processes the information to associate with the sender (as "subscriber information" associated with that sender). The service 104 may, for example, determine a percentage of times result-incorporated e-mail messages from a particular sender are blocked out of a total number of e-mail messages sent from that sender. (The service 104 also knows the number of total messages sent by that sender, since the service 104 knows how many results it generated for e-mails sent by that sender.) Similarly, the service 104 may determine a percentage of times result-incorporated e-mail messages from a particular sender are "clicked through" out of a total number of e-mail messages sent from that sender via the service 104. The information associated with the sender may be provided, for example, to receivers of result-incorporated e-mail messages. This is discussed later with reference to Figure 9.

[0067] Still referring to Figure 1 and Figure 3, in some examples, the processing at the sender's side 102, at the service 104, and at the receiver's side 106 is such to obscure, to the service 104, the identity of recipients of e-mail messages for which results are provided by the service 104. In some examples, salt technology is utilized so that the identities of the recipients are not directly provided to the service 104 from the senders' sides 104. That is, in these examples, the information regarding e-mail provided at step 112 in Figure 1 does not include the recipient e-mail address directly but, rather, provides the recipient e-mail address in a way such that the recipient e-mail address is obscured from discovery at the service.

[0068] In accordance with some examples, a hash of the recipient e-mail address is provided to the service 104 from the sender's side 102, rather than providing the recipient e-mail address in plain text. Correspondingly, in the processing on the receiver's side to assess the integrity of a received e-mail message, the hash of the recipient e-mail address is used rather than using the recipient e-mail address in plain text. See, for example, step 304 in Figure 3. Employing this processing, however, it is still relatively simple for an intruder to discern (at the service 104), for a particular known recipient, senders who have sent e-mail messages to that known recipient using the service 104. That is, one could hash the recipient e-mail address and then look for that hash result in requests sent from senders' sides 102 to the service 104. Furthermore, even if not knowingly looking for information about a particular recipient, an intruder could inspect the hash values to discern patterns of e-mail recipients sending by a particular sender (e.g., sent fifteen e-mail messages on Wednesday to a particular e-mail recipient, where the actual identity of the particular e-mail recipient is thus far unknown).

[0069] In accordance with yet other examples, the hash of the recipient address provided from the sender's side 102 to the service is not the hash of the recipient e-mail address alone but, rather, is the hash of the recipient e-mail address concatenated with "salt." Salt is a random string and, by concatenating the salt with the recipient e-mail address, less information is "leaked" from the results. Put another way, much more effort may be required to glean useful information.

[0070] In some examples, the salt is provided from the sender's side 102 to the service 104 along with the hash of the concatenated recipient e-mail address and salt pair, and the salt is included in the encrypted request that is sent back to the sender's side 102 from the service 104, for transmitting in the STMP header of an e-mail message to the

receiver's side 106. In other examples, the salt itself is not provided to the service 104 from the sender's side 102 and, thus, is not included in the encrypted request. The salt is, however, still provided to the receiver's side 106, as the salt is needed in the processing to assess the integrity of the result-incorporated e-mail message (step 120 in Figure 1).

[0071] Referring to Figure 4, it is recognized that some received e-mail messages will not be result-incorporated e-mail messages. At step 402, it is determined whether the received e-mail message is a result-incorporated e-mail message at all. This may be accomplished, for example, by checking for the appropriate SMTP header where the result would be expected to have been incorporated into the e-mail message as described above. If the e-mail message is determined to be a result incorporated e-mail message then, at step 404, processing takes place according to Figure 3. Otherwise, at step 406, appropriate action is taken. For example, preference may be to consider such unverifiable e-mail messages to be of a low priority, and a user may be given a choice of leaving the e-mail in a default inbox, or deleting it.

[0072] It is recognized that, in some instances, while the received e-mail message may be different from the sent e-mail message, the received e-mail message still is essentially incorrupt. For example, mail transport processes may legitimately change the e-mail message to add a footer such as "Do you Yahoo!?" In some examples, a "fuzzy" test is employed to determine whether the received e-mail is sound. This includes, in some examples, reversing known e-mail transport modifications from the received e-mail message before determining 302 the first digest result at the receiver's side 106.

[0073] In other examples, before the e-mail message is hashed or otherwise processed by the e-mail client at the sender 102 to provide to the service 104, features that are likely to be changed during mail transport are stripped from the e-mail message. Thus, the hash provided to the service, from which the result is determined by the service, is based on characteristics that will not be changed (or are likely to be not changed) during mail transport. Likewise, at the receiver's side, features that are likely to have been a result of change during mail transport are stripped from the received e-mail message before the first digest result is determined. In this way, changes introduced by legitimate mail transport processes are disregarded in the assessment of integrity of the received e-mail message.

[0074] In other examples, it is assumed that the meaning of the e-mail message is not legitimately changed during mail transport. Thus, the hash (or other encoding) of the e-mail message to be sent from the sender's side 102 is based on a determined meaning of the e-mail message, rather than on a literal text encoding of the e-mail message. The service 104 determines the result based on this encoding of the determined meaning. The first result is similarly determined at the receiver's side based on the determined meaning of the received e-mail message.

[0075] As discussed above, at step 306 of Figure 3, a public key associated with the service 104 (i.e., the public key associated with the private key used by the service 104 in a particular instance) is used to decode the service-determined result incorporated into the received e-mail message, to determine a decoded service-determined result. With reference to Figure 6, we now discuss how, in some examples, the appropriate public keys are accessible to the e-mail clients so that receiving e-mail clients can determine the decoded service-determined result.

[0076] In some examples, when the service 104 creates a result to provide back to the sender's side 102, to accompany the e-mail message to send to the receiver's side 106, the service 104 informs the sender's side 102 of a unique identification number for the public/private key pair associated with the determination of the result. This unique identification number is provided by the sender's side 102 to the receiver's side 106 accompanying the sent e-mail message, for example, as part of an SMTP header.

[0077] Referring specifically to Figure 6, at the receiver's side 106, the e-mail client uses the unique identification number as an index into a database 602 of certificates in a local storage 604, to determine if the database 602 already holds the appropriate public key for the e-mail client at the receiver's side 106 to determine the decoded service-determined result. If the e-mail client at the receiver's side does not already have such local access to a copy of the appropriate public key, then the e-mail client at the receiver's side sends a query 606 to the service 104 for the appropriate public key. The service provides a return message 608 including the requested public key, and the requested public key is inserted into the database 602 of the local storage 604 for subsequent use. As a result, the service is not constrained to using particular public/private key pairs, since the system is readily adaptable to changes in the public/private key pairs used by the service.

[0078] While a receiving e-mail client can somewhat easily obtain a required public key, it is desirable for each receiving e-mail client to have locally available to it the

public keys corresponding to the private keys used by the service 104 for determining the results in the result-accompanied e-mail messages. In this way, communication with the service 104 would typically not be required to process each received result-incorporated e-mail message.

[0079] Furthermore, while it is recognized that some requests to the service 104 for certificates will be necessary (e.g., when the database 602 is initially empty, and to account for expiration or other invalidation of certificates), it is desirable to spread out such requests in time. For example, if a particular certificate expires on January 1, it is undesirable for the service to be deluged with millions of requests for an updated certificate, all within a compressed time period.

[0080] In accordance with some examples, the service 104 is configured to "roll out" new certificates over a rollout time period. Accompanying a service-determined result, a notification is provided by the service 104 to the sender 102 informing of the availability of a new certificate. Based upon such a notification, the e-mail client at the sender's side 102 requests the new certificate for insertion into the database 602 of its local storage 604, for example, in a manner similar to that just discussed relative to Figure 6 with respect to requests by the receiver's side 106. The next time that e-mail client, presently a sender 102, is a receiver 106 of an e-mail message accompanied by a result encoded by the service 104 using the new certificate, the new certificate will already be locally accessible to process the result.

[0081] Collectively, a large number of the e-mail clients will have had their database 602 in local storage 604 updated, during the roll-out time period, to hold the new certificate. In some examples, the service 104 employs processing to spread out the notifications over the rollout time period such that the processing required by the service 104 to respond to the requests for updated certificates is acceptably distributed.

[0082] For those e-mail clients not updated during the roll-out time period (for example, because an e-mail client had not engaged the service 104 to send a result-accompanied e-mail message), the processing by the service 104 discussed above may be employed to provide the updated certificates on an as-needed basis. In some examples, the "new certificate" indication is passed by the sender's side 102 to the receiver's side 106, so that even those e-mail clients that are on a receiver's side 106 during the roll-out time period, but that are not on a sender's side 102 during the roll-out time period, will also request the new certificate from the service 104.

[0083] Building a critical mass of subscribers to the service 104 enhances the probability that received e-mail messages are result-incorporated e-mail messages. Referring to Figure 7, if a received e-mail message is determined to be a result-incorporated e-mail message (denoted by reference numeral 702), then step 120 (Figure 1) processing occurs. In some examples, the received email message may be accompanied by (have incorporated into it), as well as a service-determined result, an invitation to subscribe to the service 104. This situation is denoted in Figure 7 by reference numeral 704 and may be detected, for example, by inspecting a string in the SMTP header of the received e-mail message. In such a case, the received invitation e-mail message is suppressed by the e-mail client at the receiver's side 106, since the receiver is already a subscriber (as is the sender). This may include, for example, processing the invitation such that it is not presented to the user of the e-mail client at the receiver's side 106. The invitation may be automatically moved to the trash folder, or even made entirely invisible to the user of the e-mail client at the receiver's side 106.

[0084] In some examples, in the situation where the received e-mail message does not include a service-determined result (this situation is denoted by reference numeral 706), the e-mail client at the receiver's side 106 is configured to automatically send an invitation to the sender of the received e-mail message. See reference numeral 708. The invitation is sent to the "reply-to address" in the received e-mail message, through the e-mail system at the receiver's side 106. The invitation is transmitted without involvement of the service 104. In some examples, the invitation is sent only when the user at the receiver's side 106 is actually responding to the email they received without a service-supplied result. In this case, it is assumed that there is a pre-existing relationship between the parties. If invitations are sent out blindly, this could cause mail loops with some autoresponders.

[0085] In some examples, the e-mail client at the receiver's side 106 includes filter processing 710 that suppresses invitation sending under particular conditions, such as that the potential invitee is incapable of subscribing to the service (e.g., due to a particular software configuration of the sender). Another condition may be that the e-mail client at the receiver's side 106 has already caused an invitation to be sent to the particular potential invitee within some predetermined amount of time previous.

[0086] We briefly expand on the situation 704 in which an invitation is received at the receiver's side. This would typically not be a result of having been automatically sent

an invitation to a "reply-to" address (since, a priori, an invitation is not sent to a reply-to address where the received message is from a subscriber to the service 104) . Rather, for example, an invitation may have been sent to all e-mail addresses in a subscriber's contact directory (e.g., the Outlook contacts directory), without regard (or, for that matter, knowledge in some instances) as to whether the recipient is already a subscriber.

[0087] E-mail invitations sent by a "third party," such as the party providing the service 104, to non-subscribers would have a high probability of being lost, discarded or ignored, since such non-subscriber recipients will likely not recognize the "third party" sender. These e-mail invitations are also likely to be blocked by spam filters. On the other hand, if the invitation e-mails are sent to such non-subscribers directly from a subscriber's e-mail client, then these e-mail invitation messages are likely to be treated by the invitees (and the e-mail clients of the invitees) as e-mail messages coming directly from the subscriber. Thus, the e-mail invitation messages are more likely to be received and noticed. This is particularly so when the e-mail invitation messages are sent to e-mail addresses in an inviter's contact folder, since these contacts are likely to be known to the invitation recipients (and, for example, be on a "white list," from which a spam filter of the invitation recipient determines senders from whom e-mail messages should be sent through unimpeded).

[0088] As illustrated in Figure 8, in accordance with some examples, a user interacts with the service 104 using a web browser program 804. Plug-in code 806 associated with the browser program 804 operates as a bridge between Javascript code running in the browser 804 and the user's local e-mail client 802. The plug-in may operate with either a visibly executing copy of the e-mail client 802 or may launch an unobtrusive copy of the e-mail client 802 to perform the mailing. The e-mail client 802 also has associated with it plug-in code 808, which interacts with the plug-in code 806 associated with the browser program 804.

[0089] The user operating the browser program 804 may interact with the service 104 to compose invitation e-mail messages, to subscribe to the service 104. The contact information for the invitees is, for example, taken from the contact folder associated with the e-mail client, although other sources of contact information may be utilized. After the user interacts with the browser program 804 to compose the invitation e-mail messages, the browser program 804 (under the control of the associated plug-in code 806) performs a "mail merge" of the contact information (such as first name, last name

and other personalized data) into the invitation e-mail messages. The e-mail invitation messages thus may appear somewhat to have been composed manually by the user. While the e-mail invitation messages may have some features of form letters, they will still have attributes of a personally-typed e-mail (such as, for example, the manner in which the sender's name is displayed).

[0090] The invitee contact information need not be accessible to the service 104 to compose and send the e-mail invitation messages. The plug-in code associated with the browser program 804 cooperates with the plug-in code 808 associated with the e-mail client program 802 to cause the thus-composed invitation messages to be sent out via the normal e-mail channel of the user, as indicated by the arrow denoted by reference numeral 810.

[0091] Sending e-mail invitation messages in this distributed manner (i.e., by each user, rather than centrally by the service 104) protects the service 194 from bad acts on the part of a particular sender user. E-mail invitation messages are, at best, identified with the particular sender user (e.g., with the ISP of that user, or with an IP address or e-mail address associated with that user) and not with the operator of the service 104. Furthermore, as discussed above relative to e-mail invitation messages sent to a "reply-to" address, to the extent the recipients of the e-mail invitation messages have the sender user on a white list, this allows the e-mail invitation messages to bypass rules the invitees may have set to filter inbound messages, or to activate rules the invitees may have set to sort mail from trusted friends into a "high priority" folder.

[0092] We now discuss how senders 102 may initially "register" with the service 104. Generally, before the service 104 will transmit a result to the sender's side 102, it must receive an indication of agreement by the sender to terms of use of the service. The indication of agreement to terms of use of the service 104 may be as a result of an initial subscription by the sender 102 to the service. The sender's side is nominally associated with a particular e-mail address of a plurality of e-mail addresses. At the service, before transmitting the result to the sender's side, an inquiry e-mail message may be sent to the particular e-mail address, and the service may observe behavior of the sender associated with the inquiry e-mail message. For example, the expected behavior of the sender associated with the inquiry e-mail message may include the sender sending an e-mail message to the service in reply to the inquiry e-mail message. Additionally or alternately, the behavior of the sender associated with the inquiry e-mail message may include following instructions set forth in the inquiry e-mail

message, such as instructions to link to a particular universal resource locator (and, perhaps, enter some special code).. The inquiry message may be sent by a transmission channel other than the electronic mail system. For example, the transmission channel may include a hardcopy delivery service, such as a mail service or similar courier service, and may even require signature of the sender. As part of the subscription process, the subscriber provides "sender information" to the service 104, which (referring to Figure 9) is held in a database 906 of subscriber information accessible to the service 104. In addition, as discussed above, the subscriber information includes information regarding interaction by user's at the receiver's sides 106 with the result-incorporated e-mail messages sent by that subscriber as a sender.

[0093] Referring specifically to Figure 9, in some examples of the e-mail client 902, service interaction facility 904 is provided for the user to cause the e-mail client 902 to interact with the service 104, to access data associated with subscribers, from the database 906 associated with the service 104. For example, a button may be provided on a toolbar of the e-mail client 902 user interface, that can be activated (arrow 901) by the user of the e-mail client on the receiver's side 106 while viewing/selecting a result-incorporated message. As a result of activating the button, the facility 904 causes a query 908 to be sent to the service 104 requesting subscriber information for the sender.

[0094] The sender information is provided back to the e-mail client (arrow 910) and is available for display by the e-mail client 902. The sender information could be provided in simple text format or, for example, in a rich display format such as XML, or HTML. The sender information could even include, for example, a photograph of the sender if the photo was provided to the service 104 and stored in the subscriber information database 906. In addition, the processed information regarding the sender may include an indicia of "quality" (possibly represented as meter/thermometer) of the sender based on how users overall have interacted with result-incorporated e-mail messages sent by this sender (box 122 in Figure 1).

[0095] In some examples, the request for sender information (arrow 910) includes the result provided from the service 104 to the sender's side 102, and the service 104 uses the recipient e-mail address encoded into the result as a confirmation that the requester of sender information actually received an e-mail message from the sender about whom it is requesting sender information. In situations where the recipient e-mail address was provided to the service using a hash of a concatenated e-mail address/salt

pair, it is useful for the salt to have been encoded into the request (from the sender's side 102 to the service 104) and result, to minimize the opportunity for gaming the confirmation processing.

[0096] The sender information 910 provided from the service 104 via the service interaction facility 904 may include a "level of trust" associated with the identity of the sender. For example, there may be three levels of trust. The lowest level of trust would indicate only that the sender "owns" the e-mail address as established, for example, by requiring the sender to perform a click-response action on an e-mail sent to the sender's purported e-mail address by the service 104 upon subscription to the service 104. An intermediate level of trust may be established by requiring the sender to take some action based on physical mail sent to the sender's purported postal address. A highest level of trust would be established by requiring the subscriber to take yet additional actions, or even by performing checks independent of actions taken by the user specifically to establish the level of trust. For example, the industry has already adopted a formula for asserting business-class trust that is followed in getting web site certificates --- a similar process may be followed.

[0097] In some examples, an indication of the level of trust accompanies the result provided by the service 104 to the sender, to be provided from the sender's side to the receiver's side as part of the e-mail message. See, for example, the "auxiliary information" in Figure 6. At the receiver's side 106, code associated with the receiver's e-mail client then interprets the received indication of the level of trust. The ultimate source of the level of trust information is, again, the subscriber information database 906 (Figure 9) at the service 104.

[0098] Whatever the path of the level of trust information (or other information about the sender), the information may be displayed visually to the user of the e-mail client at the receiver's side 106. Other actions may be taken with respect to the level of trust such as, for example, sorting messages in the inbox associated with executing the e-mail client at the receiver's side 106 based on the indicated level of trust or color-coding or otherwise visually marking the messages or message list based on the indicated level of trust.

[0099] As another example, a facility (such as a one-click import button) may be provided to add the sender to the contact list associated with the recipient's e-mail client software (since details about the sender like name, address, phone, e-mail addresses, etc. are included in the sender information database). Support may be

provided in a format appropriate to the e-mail client, such as vcards and or the richer format supported by Outlook. (It is noted that Outlook 2003 has photo support). Links may be even be provided to web sites. As an example of the usefulness of such a facility, a real estate agent may send an email to a local prospect, and the prospect could have confidence as to the integrity of the information about the real estate agent, as provided from the sender information database. This can be a selling point for that real estate agent over other real estate agents whose information is not available from a trusted source.

[00100] As discussed above, each sender of e-mail using the service 104 provides to the service 104 information about the sender, from which the service 104 can authenticate the sender. In some examples, this information (such as, for example, an account number and password pair) is stored in the registry of the computer on which the sender's e-mail client software is being executed. It is possible that the subscriber information stored in the registry becomes out of synchronization with the corresponding information at the service 104.

[00101] Referring to Figure 10, a plug-in 1001 associated with a browser of the client seamlessly handles communication between the service 104 and the registry 1003 of the client (when the user is logged in to an account associated with the service 104) such that the subscriber information in the service 104 and in the subscriber information in the registry 1003 can be automatically synchronized.

[00102] Referring still to Figure 10, We now discuss an example of the client/service synchronization in greater detail, local client-side programming and service-based programming and data storage. In general, users can log into their account with the service 104 via the internet and make profile changes into the service 104, which will be communicated to the software running on their client computer. Frequently, as discussed above, the profile information is to be locally stored (for example, in the operating system registry 1003). Furthermore, a user may make a change from one client computer while logged into the user's account at the service from one client computer (e.g., from a client computer at work) and the change is applied to both to the client computer from which the user is logged in to make the change and also to other client computers from which the user will access the service (for example, at home).

[00103] We now describe an example process with reference to Figure 10. Each time a meaningful (i.e., one to be synchronized) change is made at the service 104 to

the user's account profile, a new random value is assigned to an Update Token for that user in the update token database 1002 at the service 104. The Update Token is a randomly-generated alphanumeric value. Using the browser plug-in component, the new value of the Update Token is poked into a known location in the user's local Windows registry of the client computer from which the user is logged into the service 104. In some examples, the alphanumeric value is not randomly generated but, rather, is a result of following a numerical sequence, to allow math compares to be performed instead of string compares, perhaps giving a greater degree of control. In addition, the current value of the update token is communicated from the service 104 to the client during each request to the service to generate a result (see arrow 1004 in Figure 10). Plug-in code on the client (in the Outlook add-in, for example, keeps track of the "last" Update Token it knows about (reference numeral 1006 in Figure 10) – referring to the last synchronization point. At strategic times, such as upon startup of the e-mail client application, or each time a request is made to the service 104 to generate a result, the plug-in programming compares the last-known Update Token 1006 with that being reported by the service 104 as the current value (arrow 1004). If the values match (compare 1008), then the client is up to date and no action is taken (1010).

[00104] If the tokens do not match, then the client acquires fresh account profile information (1012) and then updates the local copy 1006 of the update token. If the user is using multiple client computers (e.g., home and work) , each client computer will automatically synchronize to the service 104 when used.

[00105] In the course of using the service 104, users may establish a variety of personalized settings including data lists of blocked senders, which are stored on a local client computer. In response to the user operating a button on the toolbar of the e-mail client, all settings and data are copied from the client computer to an area of the service 104 associated with the user. This information can be restored at any time, either to the same client computer from which it was provided, or to a different client computer, again with a single button operation. In addition, in some examples, automatic backups of such data are provided, from the client computer to the service 104.

[00106] Referring to Figure 11, data deemed to be sensitive may be communicated between a client computer 1102 and the service 104 in a secure manner . In general, the client computer 1102 initiates the communication, and the payload data is encrypted by the client computer 1102 using a randomly-generated session key.

The session key is provided to the service 104 using a public/private key encryption scheme based on a public/private key pair maintained by the service 104 and the client (which has only the public key). A serial number associated with the public/private pair used to encrypt the session key (i.e., of which the public key is used to encrypt the session key) is provided in plain text to the service. The service determines, from the serial number, the private key to use to decrypt the session key. The service then uses the decrypted session key to decrypt the payload data.

[00107] The particular Figure 11 example process is now described. First the client decides to communicate with the service 104. The client computer chooses a random session key for data encryption (e.g., using a 128-bit session key, for an AES cipher, for block data encryption). The client uses a public key certificate (e.g., the one with the highest serial number) it has in local store/cache 1104 to encrypt the session key. This results in an envelope that can only be opened using the corresponding private key at the service 104.

[00108] The session key is "remembered" by the client 1102 in memory while the secure communication between the client computer 1102 and the service 104 takes place. The client computer 1102 encrypts the payload data using the session key, and the encrypted payload data (encrypted using the session key) is provided 1122 to the service 104. The encrypted session key and a plain-text field representing the serial number of the certificate used to encrypt the session key are also provided 1120 to the service 104. This may be done securely without resorting to HTTPS or other high overhead secure communication schemes.

[00109] The service receives the request and inspects the value of the serial number (transmitted to the service 104 in plain text) used to create the payload data. Referring to the certificate store 1108 at the server, if the serial number is within an acceptable range (meaning, one that is valid to use, as opposed to one that may have been revoked or expired), the private key corresponding to that serial number is returned from the certificate server 1108 and is used to decrypt the session key 1110. The decrypted session key is then used to decode 1112 the -encrypted data provided 1122 from the client computer 1102 to the service 104.

[00110] The service 104 uses the same session key to encrypt 112 data to go to the client 1102 (note the bidirectional arrows on line 1122). If the service 104 were to generate a new session key and send it to the client in an envelope created with the private key, this would leave a security hole open – since hackers armed with the

corresponding public key could open the envelope. Since the only place the session key is stored is within the memory of the client computer 1102, which is inaccessible to outsiders, the implementation is safe and secure.

[00111] In the event the client 1102 creates an envelope using a serial number that is no longer acceptable to the service 104, the service will reject 1124 the request without further inspection. The client 1102 will detect the "bad serial number" rejection indicator 1124 and resubmit 1126 the request using a different certificate. For example, the client computer 1102 may query the service for an updated certificate, then resubmit the request using the new certificate, the entire interaction taking place without user intervention.

[00112] This use of certificates does not utilize the element of "time" as a factor for determining whether a certificate is valid or otherwise "usable" and, thus, a client computer 1102 not having the correct time/date settings does not affect the data transfer process. As discussed above, certificates can naturally expire or be revoked and the net effect is that a client computer 1102 synchronizes with the service 104 automatically on the next work request. The service 104 gets to decide which certificates it chooses to accept.

[00113] While we have described particular illustrated embodiments, it will be appreciated that various alterations, modifications and adaptations may be based on the present disclosure, and are intended to be within the scope of the present invention. While the invention has been described in connection with what are presently considered to be the most practical and preferred embodiments, it is to be understood that the present invention is not limited to the disclosed embodiment but, on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the claims.